A Fast Distributed Solver for Symmetric Diagonally Dominant Linear Equations

Rasul Tutunov[†], Haitham Bou Ammar[†], Ali Jadbabaie[♯]

{*tutunov, haithamb, jadbabai*} *@seas.upenn.edu*
† : *University of Pennsylvania, Computer and Information Science Dept.*
Levine Hall 3330 Walnut Street Philadelphia, PA 19104-6309
♯: *University of Pennsylvania, Dept. of Electrical and System Engineering*
Levine Hall 3330 Walnut Street Philadelphia, PA 19104-6309

Abstract

In this paper, we propose a fast distributed solver for linear equations given by symmetric diagonally dominant M-Matrices. Our approach is based on a distributed implementation of the parallel solver of Spielman and Peng by considering a specific approximated inverse chain which can be computed efficiently in a distributed fashion. Representing the system of equations by a graph $\mathbb{G}$, the proposed distributed algorithm is capable of attaining ϵ-close solutions (for arbitrary ϵ) in time propotional to n^3 (number of nodes in $\mathbb{G}$), $\boldsymbol{\alpha}$ (upper bound on the size of the R-Hop neighborhood), and $\frac{\boldsymbol{W}_{\max}}{\boldsymbol{W}_{\min}}$ (maximum and minimum weights of edges in $\mathbb{G}$).

arXiv:1502.03158v1 [cs.DC] 11 Feb 2015

1. Introduction

Solving systems of linear equations in symmetric diagonally matrices (SDD) is of interest to researchers in a variety of fields including but not limited to, solutions to partial differential equations [7], computations of maximum flows in graphs [9], machine learning [23], and as basis for various algorithms [8].

Much interest has been devoted to determining fast algorithms for solving SDD systems. Spielman and Teng [21] proposed a nearly linear-time algorithm for solving SDD systems, which benefited from the multi-level framework of [1, 10], preconditioners [6], and spectral graph sparsifiers [3, 22]. Further exploiting these ingredients, Koutis *et. al* [14, 15] developed an even faster algorithm for acquiring ϵ-close solutions to SDD linear systems. Further improvements have been discovered by Kelner *et. al* [12], where their algorithm relied on only spanning-trees and eliminated the need for graph sparsifiers and the multi-level framework.

On the parallel side, much progress has been made on developing such solvers. Koutis and Miller [13] proposed an algorithm requiring nearly-linear work and $m^{1/6}$ depth for planar graphs. This was then extended to general graphs by [5] leading to depth close to $m^{1/3}$. Peng and Spielman [20] have proposed an efficient parallel solver requiring nearly-linear work and poly-logarithmic depth without the need for low-stretch spanning trees. Their algorithm, which we distribute in this paper, requires sparse approximate inverse chains [20] which facilitates the solution of the SDD system.

Less progress, on the other hand, has been made on the distributed version of these solvers. Current methods, e.g., Jacobi iteration [2, 4], can be used for distributed solutions but require substantial complexity. In [17], the authors propose a gossiping framework which can be used for a distributed solution of the above linear system. Recent work [16] considers a local and asynchronous solution for solving systems of linear equations, where they acquire a bound on the number of needed multiplication proportional to the degree and condition number for one component of the solution vector.

Contributions: In this paper, we propose a fast distributed solver for linear equations given by symmetric diagonally dominant M-Matrices. Our approach distributes the parallel solver in [20] by considering a specific approximated inverse chain which can be computed efficiently in a distributed fashion. Our algorithm's computational complexity is given by $\mathcal{O}\left(n^3 \frac{\alpha}{R} \frac{W_{\max}}{W_{\min}} \log\left(\frac{1}{\epsilon}\right)\right)$, with n being the number of nodes in graph $\mathbb{G}$, $W_{\max}$ and $W_{\min}$ denoting the largest and smaller weights of the edges in $\mathbb{G}$, respectively, $\alpha = \min\left\{n, \frac{d_{\max}^{R+1}-1}{d_{\max}-1}\right\}$ representing the upper bound on the size of the R-Hop neighborhood $\forall v \in \mathbb{V}$, and $\epsilon \in (0, \frac{1}{2}]$ being the precision parameter. Our approach improves current linear methods by a factor of $\log n$ and by a factor of the degree compared to [16] for each component of the solution vector.

2. Problem Definition & Notation

We consider the following system of linear equations:

$$M_0 x = b_0 \tag{1}$$

where M_0 is a Symmetric Diagonally Dominant M-Matrix (SDDM). Namely, M_0 is symmetric positive definite with non-positive off diagonal elements, such that for all $i = 1, 2, \ldots, n$:

$$[M_0]_{ii} \geq -\sum_{j=1, j\neq i}^{n} [M_0]_{ij}$$

The system of Equations in 1 can be interpreted as representing an undirected weighted graph, $\mathbb{G}$, with M_0 being its Laplacian. Namely, $\mathbb{G} = (\mathbb{V}, \mathbb{E}, W)$, with $\mathbb{V}$ representing the set of nodes, $\mathbb{E}$ denoting the edges, and W representing the weighted graph adjacency. Nodes v_i and v_j are connected with an edge $e = (i, j)$ iff $W_{ij} > 0$, where:

$$W_{ij} = [M_0]_{ii} \quad (\text{if } i = j), \quad \text{or} \quad W_{ij} = -[M_0]_{ij}, \quad \text{otherwise.}$$

Following [20], we seek ϵ-approximate solutions to $x^\star$, being the exact solution of $M_0 x = b_0$, defined as:

Definition 1. Let $x^\star \in \mathbb{R}^n$ be the solution of $Mx = b_0$. A vector $\tilde{x} \in \mathbb{R}^n$ is called an $\epsilon-$ approximate solution, if:

$$||x^\star - \tilde{x}||_{M_0} \leq \epsilon ||x^\star||_{M_0}, \quad \text{where } ||u||_{M_0}^2 = u^\top M_0 u. \tag{2}$$

The R-hop neighbourhood of node $\boldsymbol{v}_k$ is defined as $\mathbb{N}_r\left(\boldsymbol{v}_k\right) = \{\boldsymbol{v} \in \mathbb{V} : \text{dist}\left(\boldsymbol{v}_k, \boldsymbol{v}\right) \leq r\}$. We also make use of the diameter of a graph, $\mathbb{G}$, defined as $\text{diam}\left(\mathbb{G}\right) = \max_{\boldsymbol{v}_i, \boldsymbol{v}_j \in \mathbb{V}} \text{dist}\left(\boldsymbol{v}_i, \boldsymbol{v}_j\right)$.

Definition 2. We say that a matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ has a sparsity pattern corresponding to the R-hop neighborhood if $\boldsymbol{A}_{ij} = 0$ for all $i = 1, \ldots, n$ and for all j such that $\boldsymbol{v}_j \notin \mathbb{N}_r\left(\boldsymbol{v}_i\right)$.

We will denote the spectral radius of a matrix $\boldsymbol{A}$ by $\rho\left(\boldsymbol{A}\right) = \max |\boldsymbol{\lambda}_i|$, where $\boldsymbol{\lambda}_i$ represents an eigenvalue of the matrix $\boldsymbol{A}$. Furthermore, we will make use of the condition number[1], $\kappa\left(\boldsymbol{A}\right)$ of a matrix $\boldsymbol{A}$ defined as $\kappa = \left|\frac{\boldsymbol{\lambda}_{\max}(\boldsymbol{A})}{\boldsymbol{\lambda}_{\min}(\boldsymbol{A})}\right|$. In [21] it is shown that the condition number of the graph Laplacian is at most $\mathcal{O}\left(n^3 \frac{\boldsymbol{W}_{\max}}{\boldsymbol{W}_{\min}}\right)$, where $\boldsymbol{W}_{\max}$ and $\boldsymbol{W}_{\min}$ represent the largest and the smallest edge weights in $\mathbb{G}$. Finally, the condition number of a sub-matrix of the Laplacian is at most $\mathcal{O}\left(n^4 \frac{\boldsymbol{W}_{\max}}{\boldsymbol{W}_{\min}}\right)$, see [20].

2.1. Problem Definition

We assume that each node $\boldsymbol{v}_k \in \mathbb{V}$ has information about the weights of adjacent edges. Further, each node $\boldsymbol{v}_k$ has the capabilities of storing the value of the ith component of $\boldsymbol{b}_0$, which is denoted as $[\boldsymbol{b}_0]_i$. At each time step, nodes can exchange information with their neighboors. Each node is responsible for determining the corresponding component, $\boldsymbol{x}_i$, of the solution vector $\boldsymbol{x} \in \mathbb{R}^n$. We also assume a synchronized model whereby time complexity is measured by a global clock. The **goal** is to find ϵ-approximate solution for $\boldsymbol{M}_0 \boldsymbol{x} = \boldsymbol{b}_0$ in a *distributed fashion*, while being restricted to R-hop communication between the nodes.

3. Background

3.1. Standard Splittings & Approximations

Following the setup in [20], we provide standard definitions required in the remainder of the paper:

Definition 3. The standard splitting of a symmetric matrix $\boldsymbol{M}_0$ is:

$$\boldsymbol{M}_0 = \boldsymbol{D}_0 - \boldsymbol{A}_0 \tag{3}$$

here $\boldsymbol{D}_0$ is a diagonal matrix such that $[\boldsymbol{D}_0]_{ii} = [\boldsymbol{M}_0]_{ii}$ for $i = 1, 2, \ldots, n$, and $\boldsymbol{A}_0$ representing a non-negative symmetric matrix such that $[\boldsymbol{A}_0]_{ij} = -[\boldsymbol{M}_0]_{ij}$ if $i \neq j$, and $[\boldsymbol{A}_0]_{ii} = 0$.

We also define the Loewner ordering:

Definition 4. Let $\mathcal{S}_{(n)}$ be the space of $n \times n$-symmetric matrices. The Loewner ordering $\preceq$ is a partial order on $\mathcal{S}_{(n)}$ such that $\boldsymbol{Y} \preceq \boldsymbol{X}$ if and only if $\boldsymbol{X} - \boldsymbol{Y}$ is positive semidefinite.

Finally, we define the "$\approx_\alpha$" operation used in the sequel to come as:

Definition 5. Let $\boldsymbol{X}$ and $\boldsymbol{Y}$ be positive semidefinite symmetric matrices. Then $\boldsymbol{X} \approx_\alpha \boldsymbol{Y}$ if and only iff

$$e^{-\alpha}\boldsymbol{X} \preceq \boldsymbol{Y} \preceq e^\alpha \boldsymbol{X} \tag{4}$$

with $\boldsymbol{A} \preceq \boldsymbol{B}$ meaning $\boldsymbol{B} - \boldsymbol{A}$ is positive semidefinite.

Based on the above definitions, the following lemma represents the basic characteristics of the $\approx_\alpha$ operator:

Lemma 1. *[20] Let $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$ and, $\boldsymbol{Q}$ be symmetric positive semi definite matrices. Then*

(1) If $\boldsymbol{X} \approx_\alpha \boldsymbol{Y}$, then $\boldsymbol{X} + \boldsymbol{Z} \approx_\alpha \boldsymbol{Y} + \boldsymbol{Z}$, (2) If $\boldsymbol{X} \approx_\alpha \boldsymbol{Y}$ and $\boldsymbol{Z} \approx_\alpha \boldsymbol{Q}$, then $\boldsymbol{X} + \boldsymbol{Z} \approx_\alpha \boldsymbol{Y} + \boldsymbol{Q}$
(3) If $\boldsymbol{X} \approx_\alpha \boldsymbol{Y}$ and $\boldsymbol{Z} \approx_\alpha \boldsymbol{Q}$, then $\boldsymbol{X} + \boldsymbol{Z} \approx_\alpha \boldsymbol{Y} + \boldsymbol{Q}$, (4) If $\boldsymbol{X} \approx_{\alpha_1} \boldsymbol{Y}$ and $\boldsymbol{Y} \approx_{\alpha_2} \boldsymbol{Z}$, then $\boldsymbol{X} \approx_{\alpha_1 + \alpha_2} \boldsymbol{Z}$
(5) If $\boldsymbol{X}$, and $\boldsymbol{Y}$ are non singular and $\boldsymbol{X} \approx_\alpha \boldsymbol{Y}$, then $\boldsymbol{X}^{-1} \approx_\alpha \boldsymbol{Y}^{-1}$, (6) If $\boldsymbol{X} \approx_\alpha \boldsymbol{Y}$ and $\boldsymbol{V}$ is a matrix, then $\boldsymbol{V}^\mathsf{T}\boldsymbol{X}\boldsymbol{V} \approx_\alpha \boldsymbol{V}^\mathsf{T}\boldsymbol{Y}\boldsymbol{V}$

The next lemma shows that good approximations of $\boldsymbol{M}_0^{-1}$ guarantee good approximated solutions of $\boldsymbol{M}_0 \boldsymbol{x} = \boldsymbol{b}_0$.

Lemma 2. *Let $\boldsymbol{Z}_0 \approx_\epsilon \boldsymbol{M}_0^{-1}$, and $\tilde{\boldsymbol{x}} = \boldsymbol{Z}_0 \boldsymbol{b}_0$. Then $\tilde{\boldsymbol{x}}$ is $\sqrt{2^\epsilon(e^\epsilon - 1)}$ approximate solution of $\boldsymbol{M}_0 \boldsymbol{x} = \boldsymbol{b}_0$.*

Proof. The proof can be found in the appendix. $\square$

We next discuss the parallel SDDM solver introduced in [20].

---

[1] Please note that in the case of the graph Laplacian, the condition number is defined as the ratio of the largest to the smallest nonzero eigenvalues.

3.2. The Parallel SDDM Solver

The parallel SDDM solver proposed in [20] is a parallelized technique for solving the problem of Section 2.1. It makes use of inverse approximated chains (see Definition 6) to determine $\tilde{x}$ and can be split in two steps. In the first step, denoted as Algorithm 1, a "crude" approximation, x_0, of $\tilde{x}$ is returned. x_0 is driven to the ϵ-close solution, $\tilde{x}$, using Richardson Preconditioning in Algorithm 2. Before we proceed, we start with the following two Lemmas which enable the definition of inverse chain approximation.

Lemma 3. *[20] If $M = D - A$ is an SDDM matrix, with D being positive diagonal, and A denoting a non-negative symmetric matrix, then $D - AD^{-1}A$ is also SDDM.*

Lemma 4. *[20] Let $M = D - A$ be an SDDM matrix, where D is positive diagonal and, A a symmetric matrix. Then*

$$(D - A)^{-1} = \frac{1}{2}\Big[D^{-1} + \big(I + D^{-1}A\big)\big(D - AD^{-1}A\big)^{-1}\big(I + AD^{-1}\big)\Big] \tag{5}$$

Given the results in Lemmas 3 and 4, we now can consider inverse approximated chains of M_0:

Definition 6. Let $\mathcal{C} = \{M_0, M_1, \dots, M_d\}$ be a collection of SDDM matrices such that $M_i = D_i - A_i$, with D_i a positive diagonal matrix, and A_i denoting a non-negative symmetric matrix. Then $\mathcal{C}$ is an inverse approximated chain if there exists positive real numbers $\epsilon_0, \epsilon_1, \dots, \epsilon_d$ such that: (1) For $i = 1, \dots, d$: $D_i - A_i \approx_{e_{i-1}} D_{i-1} - A_{i-1}D_{i-1}^{-1}A_{i-1}$, (2) For $i = 1, \dots, d$: $D_i \approx_{\epsilon_{i-1}} D_{i-1}$, and (3) $D_d \approx_{\epsilon_d} D_d - A_d$.

Algorithm 1 ParallelRSolve $(M_0, M_1, \dots, M_d, b_0)$

1: **Input**: Inverse approximated chain, $\{M_0, M_1, \dots, M_d\}$, and b_0 being
2: **Output**: The "crude" approximation, x_0, of $x^\star$
3: **for** $i = 1$ to d **do**
4: $\quad b_i = \big(I + A_{i-1}D_{i-1}^{-1}\big)b_{i-1}$
   end for
5: $x_d = D_d^{-1}b_d$
6: **for** $i = d - 1$ to 0 **do**
7: $\quad x_i = \frac{1}{2}\big[D_i^{-1}b_i + \big(I + D_i^{-1}A_i\big)x_{i+1}\big]$
   end for
8: **return** x_0

The quality of the "crude" solution returned by Algorithm 1 is quantified in the following lemma:

Lemma 5. *[20] Let $\{M_0, M_1, \dots, M_d\}$ be the inverse approximated chain and denote Z_0 be the operator defined by ParallelRSolve $(M_0, M_1, \dots, M_d, b_0)$, namely, $x_0 = Z_0 b_0$. Then*

$$Z_0 \approx_{\sum_{i=0}^{d} \epsilon_i} M_0^{-1} \tag{6}$$

Algorithm 1 returns a "crude" solution to $M_0 x = b$. To obtain arbitrary close solutions, Spielman *et.al* [20] introduced the *preconditioned Richardson iterative scheme*, summarized in Algorithm 2. Following their analysis, Lemma 6 provides the iteration count needed by Algorithm 2 to arrive at $\tilde{x}$.

Lemma 6. *[20] Let $\{M_0, M_1 \dots M_d\}$ be an inverse approximated chain such that $\sum_{i=1}^{d} \epsilon_i < \frac{1}{3}\ln 2$. Then ParallelESolve $(M_0, M_1, \dots, M_d, b_0, \epsilon)$ arrives at an ϵ close solution of $x^\star$ in $q = \mathcal{O}\big(\log \frac{1}{\epsilon}\big)$ iterations.*

4. Distributed SDDM Solvers

Next, we distribute the parallel solver of Section 3.2. Similar to [20], we first introduce an approximate inverse chain which can be computed in a distributed fashion. This leads us to distributed version of the "crude" solver (see Section 4.1). Contrary to [20], however, we then generalize the "crude" distributed solver to allow for *exact* solutions (see Section 4.2) of Equation 1. We summarize our results in the following theorem:

Theorem 1. *There exists a distributed algorithm, $\mathcal{A}\big(\{[M_0]_{k1}, \dots [M_0]_{kn}\}, [b_0]_k, \epsilon\big)$, that computes ϵ-close approximations to the solution of $M_0 x = b_0$ in $\mathcal{O}\big(n^2 \log \kappa \log\big(\frac{1}{\epsilon}\big)\big)$ time steps, with n the number of nodes in $\mathbb{G}$, κ the condition number of M_0, and $[M_0]_k$. the kth row of M_0, as well as $\epsilon \in \big(0, \frac{1}{2}\big]$ representing the precision parameter.*

Note that for each node $v_k \in \mathbb{V}$, the input information for algorithm $\mathcal{A}$ is the kth row of M_0 (i.e., the weights of the edges adjacent to v_i), the precision parameter ϵ, and the kth component of $[b_0]$ (i.e., $[b_0]_k$), easily rendering a distributed solver.

Algorithm 2 ParallelESolve $(\boldsymbol{M}_0, \boldsymbol{M}_1, \ldots, \boldsymbol{M}_d, \boldsymbol{b}_0, \epsilon)$

1: **Input**: Inverse approximated chain $\{\boldsymbol{M}_0, \boldsymbol{M}_1, \ldots, \boldsymbol{M}_d\}$, $\boldsymbol{b}_0$, and ϵ.
2: **Output**: ϵ close approximation, $\tilde{\boldsymbol{x}}$, of $\boldsymbol{x}^*$
3: **Initialize**: $\boldsymbol{y}_0 = 0$;
4: $\chi = \text{ParallelRSolve}\,(\boldsymbol{M}_0, \boldsymbol{M}_1, \ldots, \boldsymbol{M}_d, \boldsymbol{b}_0)$ (i.e., Algorithm 1)
5: **for** $k = 1$ to q **do**
6: $\qquad \boldsymbol{u}_k^{(1)} = \boldsymbol{M}_0 \boldsymbol{y}_{k-1}$
7: $\qquad \boldsymbol{u}_k^{(2)} = \text{ParallelRSolve}\left(\boldsymbol{M}_0, \boldsymbol{M}_1, \ldots, \boldsymbol{M}_d, \boldsymbol{u}_k^{(1)}\right)$
8: $\qquad \boldsymbol{y}_k = \boldsymbol{y}_{k-1} - \boldsymbol{u}_k^{(2)} + \chi$
$\quad$ **end for**
9: $\tilde{\boldsymbol{x}} = \boldsymbol{y}_q$
10: **return** $\tilde{\boldsymbol{x}}$

Algorithm 3 DistrRSolve $\left(\left\{[\boldsymbol{M}_0]_{k1}, \ldots, [\boldsymbol{M}_0]_{kn}\right\}, [\boldsymbol{b}_0]_k, d\right)$

Part One: Computing $[\boldsymbol{b}_i]_k$
$[\boldsymbol{b}_1]_k = [\boldsymbol{b}_0]_k + \sum_{j : \boldsymbol{v}_j \in \mathbb{N}_1(\boldsymbol{v}_k)} [\boldsymbol{A}_0 \boldsymbol{D}_0^{-1}]_{kj} [\boldsymbol{b}_0]_j$
for $i = 2$ to d **do**
$\quad$ **for** $j : \boldsymbol{v}_j \in \mathbb{N}_{2^{i-1}}(\boldsymbol{v}_k)$ **do**
$\qquad \left[(\boldsymbol{A}_0 \boldsymbol{D}_0^{-1})^{2^{i-1}}\right]_{kj} = \sum_{r=1}^n \frac{[\boldsymbol{D}_0]_{rr}}{[\boldsymbol{D}_0]_{jj}} \left[(\boldsymbol{A}_0 \boldsymbol{D}_0^{-1})^{2^{i-2}}\right]_{kr} \left[(\boldsymbol{A}_0 \boldsymbol{D}_0^{-1})^{2^{i-2}}\right]_{jr}$
$\quad$ **end for**
$\quad [\boldsymbol{b}_i]_k = [\boldsymbol{b}_{i-1}]_k + \sum_{j : \boldsymbol{v}_j \in \mathbb{N}_{2^{i-1}}(\boldsymbol{v}_k)} \left[(\boldsymbol{A}_0 \boldsymbol{D}_0^{-1})^{2^{i-1}}\right]_{kj} [\boldsymbol{b}_{i-1}]_j$
end for

Part Two: Computing $[\boldsymbol{x}_0]_k$
$[\boldsymbol{x}_d]_k = [\boldsymbol{b}_d]_k / [\boldsymbol{D}_0]_{kk}$
for $i = d-1$ to 1 **do**
$\quad$ **for** $j : \boldsymbol{v}_j \in \mathbb{N}_{2^i}(\boldsymbol{v}_k)$ **do**
$\qquad \left[(\boldsymbol{D}_0^{-1} \boldsymbol{A}_0)^{2^i}\right]_{kj} = \sum_{r=1}^n \frac{[\boldsymbol{D}_0]_{jj}}{[\boldsymbol{D}_0]_{rr}} \left[(\boldsymbol{D}_0^{-1} \boldsymbol{A}_0)^{2^{i-1}}\right]_{kr} \left[(\boldsymbol{D}_0^{-1} \boldsymbol{A}_0)^{2^{i-1}}\right]_{jr}$
$\quad$ **end for**
$\quad [\boldsymbol{x}_i]_k = \frac{[\boldsymbol{b}_i]_k}{2[\boldsymbol{D}_0]_{kk}} + \frac{[\boldsymbol{x}_{i+1}]_{k+1}}{2} + \frac{1}{2} \sum_{j : \boldsymbol{v}_j \in \mathbb{N}_{2^i}(\boldsymbol{v}_k)} \left[(\boldsymbol{D}_0^{-1} \boldsymbol{A}_0)^{2^i}\right]_{kj} [\boldsymbol{x}_{i+1}]_j$
end for
$[\boldsymbol{x}_0]_k = \frac{[\boldsymbol{b}_0]_k}{2[\boldsymbol{D}_0]_{kk}} + \frac{[\boldsymbol{x}_1]_k}{2} + \frac{1}{2} \sum_{j : \boldsymbol{v}_j \in \mathbb{N}_1(\boldsymbol{v}_k)} [\boldsymbol{D}_0^{-1} \boldsymbol{A}_0]_{kj} [\boldsymbol{x}_1]_j$
return: $[\boldsymbol{x}_0]_k$

4.1. "Crude" Distributed SDDM Solver

Starting from $\boldsymbol{M}_0 = \boldsymbol{D}_0 - \boldsymbol{A}_0$, consider the collection $\mathcal{C} = \{\boldsymbol{A}_0, \boldsymbol{D}_0, \boldsymbol{A}_1, \boldsymbol{D}_1, \ldots, \boldsymbol{A}_d, \boldsymbol{D}_d\}$, where $\boldsymbol{D}_k = \boldsymbol{D}_0$, and $\boldsymbol{A}_k = \boldsymbol{D}_0 \left(\boldsymbol{D}_0^{-1} \boldsymbol{A}_0\right)^{2^k}$, for $k = \{1, \ldots, d\}$ with $\boldsymbol{D}_0 = \boldsymbol{D}_0$, and $\boldsymbol{A}_0 = \boldsymbol{A}_0$. Since the magnitude of the eigenvalues of $\boldsymbol{D}_0^{-1} \boldsymbol{A}_0$ is strictly less than 1, $\left(\boldsymbol{D}_0^{-1} \boldsymbol{A}_0\right)^{2^k}$ tends to zero as k increases which reduces the length of the chain needed for the distributed solver. It is easy to verify that $\mathcal{C}$ is an inverse approximated chain, since: (1) $\boldsymbol{D}_i - \boldsymbol{A}_i \approx_{\epsilon_{i-1}} \boldsymbol{D}_{i-1} - \boldsymbol{A}_{i-1} \boldsymbol{D}_{i-1}^{-1} \boldsymbol{A}_{i-1}$ with $\epsilon_i = 0$ for $i = 1, \ldots, d$, (2) $\boldsymbol{D}_i \approx_{\epsilon_{i-1}} \boldsymbol{D}_{i-1}$ with $\epsilon_i = 0$ for $i = 1, \ldots, d$, and (3) $\boldsymbol{D}_d \approx_{\epsilon_d} \boldsymbol{D}_d - \boldsymbol{A}_d$. Using the above, Algorithm 3 (our first contribution) describes the distributed version of the "crude" parallel solver, which returns the kth component of the approximate solution vector, $[\boldsymbol{x}_0]_k$. Each node, $\boldsymbol{v}_k \in \mathbb{V}$, receives the kth row of $\boldsymbol{M}_0$, the kth value of $\boldsymbol{b}_0$ (i.e., $[\boldsymbol{b}_0]_k$), and the length of the inverse approximated chain d as inputs. It operates in two parts, **Part One** and **Part Two**. The first, computes the kth component of $\boldsymbol{b}_i$, $[\boldsymbol{b}_i]_k$ for $i \in \{1, \ldots, d\}$, which is then used in **Part Two** to return $[\boldsymbol{x}_0]_k$.

Analysis of Algorithm 3: Next, we present the theoretical analysis, showing that DistrRSolve computes the kth component of the "crude" approximation of $\boldsymbol{x}^\star$. Further, we provide the time complexity analysis.

Lemma 7. *Let* $\boldsymbol{M}_0 = \boldsymbol{D}_0 - \boldsymbol{A}_0$ *be the standard splitting of* $\boldsymbol{M}_0$. *Let* $\boldsymbol{Z}_0'$ *be the operator defined by*

Algorithm 4 DistrESolve $(\{[\boldsymbol{M}_0]_{k1}, \ldots, [\boldsymbol{M}_0]_{kn}\}, [\boldsymbol{b}_0]_k, d, \epsilon)$

    Initialize: $[\boldsymbol{y}_0]_k = 0$; $[\chi]_k = $ DistrRSolve $(\{[\boldsymbol{M}_0]_{k1}, \ldots, [\boldsymbol{M}_0]_{kn}\}, [\boldsymbol{b}_0]_k, d)$ (i.e., Algorithm 3)
    for $t = 1$ to q **do**
        $\left[\boldsymbol{u}_t^{(1)}\right]_k = [\boldsymbol{D}_0]_{kk}[\boldsymbol{y}_{t-1}]_k - \sum_{j : \boldsymbol{v}_j \in \mathbb{N}_1(\boldsymbol{v}_k)}[\boldsymbol{A}_0]_{kj}[\boldsymbol{y}_{t-1}]_j$
        $\left[\boldsymbol{u}_t^{(2)}\right]_k = $ DistrRSolve $(\{[\boldsymbol{M}_0]_{k1}, \ldots, [\boldsymbol{M}_0]_{kn}\}, \left[\boldsymbol{u}_t^{(1)}\right]_k, d,)$
        $[\boldsymbol{y}_t]_k = [\boldsymbol{y}_{t-1}]_k - \left[\boldsymbol{u}_t^{(2)}\right]_k + [\chi]_k$
    end for
    $[\tilde{\boldsymbol{x}}]_k = [\boldsymbol{y}_q]_k$
    return $[\tilde{\boldsymbol{x}}]_k$

$DistrRSolve([\{[\boldsymbol{M}_0]_{k1}, \ldots, [\boldsymbol{M}_0]_{kn}\}, [\boldsymbol{b}_0]_k, d)$ *(i.e.,* $\boldsymbol{x}_0 = \boldsymbol{Z}_0' \boldsymbol{b}_0$*). Then*

$$\boldsymbol{Z}_0' \approx_{\epsilon_d} \boldsymbol{M}_0^{-1}$$

Moreover, Algorithm 3 requires $\mathcal{O}\left(dn^2\right)$ *time steps.*

Proof. See Appendix.

$\square$

4.2. "Exact" Distributed SDDM Solver

    Having introduced DistrRSolve, we are now ready to present a distributed version of Algorithm 2 which enables the computation of ϵ close solutions for $\boldsymbol{M}_0 \boldsymbol{x} = \boldsymbol{b}_0$. Similar to DistrRSolve, each node $\boldsymbol{v}_k \in \mathbb{V}$ receives the kth row of $\boldsymbol{M}_0$, $[\boldsymbol{b}_0]_k$, d and a precision parameter ϵ as inputs. Node $\boldsymbol{v}_k$ then computes the kth component of the ϵ close approximation of $\boldsymbol{x}^\star$.

    Analysis of Algorithm 4: The following lemma shows that DistrESolve computes the kth component of the ϵ close approximation of $x^\star$ and provides the time complexity analysis

Lemma 8. *Let* $\boldsymbol{M}_0 = \boldsymbol{D}_0 - \boldsymbol{A}_0$ *be the standard splitting. Further, let* $\epsilon_d < \frac{1}{3} \ln 2$ *in the nverse approximated chain* $\mathcal{C} = \{\boldsymbol{A}_0, \boldsymbol{D}_0, \boldsymbol{A}_1, \boldsymbol{D}_1, \ldots, \boldsymbol{A}_d, \boldsymbol{D}_d\}$. *Then* $DistrESolve(\{[\boldsymbol{M}_0]_{k1}, \ldots, [\boldsymbol{M}_0]_{kn}\}, [\boldsymbol{b}_0]_k, d, \epsilon)$ *requires* $\mathcal{O}\left(\log \frac{1}{\epsilon}\right)$ *iterations to return the* k^{th} *component of the* ϵ *close approximation for* $x^\star$.

Proof. See Appendix.

$\square$

    The following lemma provides the time complexity analysis of DistrESolve:

Lemma 9. *Let* $\boldsymbol{M}_0 = \boldsymbol{D}_0 - \boldsymbol{A}_0$ *be the standard splitting. Further, let* $\epsilon_d < \frac{1}{3} \ln 2$ *in the inverse approximated chain* $\mathcal{C} = \{\boldsymbol{A}_0, \boldsymbol{D}_0, \boldsymbol{A}_1, \boldsymbol{D}_1, \ldots, \boldsymbol{A}_d, \boldsymbol{D}_d\}$. *Then,* $DistrESolve(\{[\boldsymbol{M}_0]_{k1}, \ldots, [\boldsymbol{M}_0]_{kn}\}, [\boldsymbol{b}_0]_k, d, \epsilon)$ *requires* $\mathcal{O}\left(dn^2 \log(\frac{1}{\epsilon})\right)$ *time steps.*

Proof. See Appendix.

$\square$

4.3. Length of the Inverse Chain

    Both introduced algorithms depend on the length of the inverse approximated chain, d. Here, we provide an analysis to determine the value of d which guarantees $\epsilon_d < \frac{1}{3} \ln 2$ in $\mathcal{C} = \{\boldsymbol{A}_0, \boldsymbol{D}_0, \boldsymbol{A}_1, \boldsymbol{D}_1, \ldots, \boldsymbol{A}_d, \boldsymbol{D}_d\}$:

Lemma 10. *Let* $\boldsymbol{M}_0 = \boldsymbol{D}_0 - \boldsymbol{A}_0$ *be the standard splitting and* κ *denote the condition number of* $\boldsymbol{M}_0$. *Consider the inverse approximated chain* $\mathcal{C} = \{\boldsymbol{A}_0, \boldsymbol{D}_0, \boldsymbol{A}_1, \boldsymbol{D}_1, \ldots, \boldsymbol{A}_d, \boldsymbol{D}_d\}$ *with a length* $d = \lceil \log\left(2 \ln\left(\frac{\sqrt[3]{2}}{\sqrt[3]{2}-1}\right) \kappa\right) \rceil$, *then* $\boldsymbol{D}_0 \approx_{\epsilon_d} \boldsymbol{D}_0 - \boldsymbol{D}_0 \left(\boldsymbol{D}_0^{-1} \boldsymbol{A}_0\right)^{2^d}$, *with* $\epsilon_d < \frac{1}{3} \ln 2$.

Proof. The proof will be given as a collection of claims:
<u>Claim:</u> Let κ be the condition number of $\boldsymbol{M}_0 = \boldsymbol{D}_0 - \boldsymbol{A}_0$, and $\{\lambda_i\}_{i=1}^n$ denote the eigenvalues of $\boldsymbol{D}_0^{-1} \boldsymbol{A}_0$. Then, $|\lambda_i| \le 1 - \frac{1}{\kappa}$, for all $i = 1, \ldots, n$

Proof. See Appendix.

$\square$

Notice that if λ_i represented an eigenvalue of $\boldsymbol{D}_0^{-1}\boldsymbol{A}_0$, then λ_i^r is an eigenvalue of $\left(\boldsymbol{D}_0^{-1}\boldsymbol{A}_0\right)^r$ for all $r \in \mathbb{N}$. Therefore, we have

$$\rho\left(\left(\boldsymbol{D}_0^{-1}\boldsymbol{A}_0\right)^{2^d}\right) \leq \left(1 - \frac{1}{\kappa}\right)^{2^d} \tag{7}$$

<u>Claim:</u> Let $\boldsymbol{M}$ be an SDDM matrix and consider the splitting $\boldsymbol{M} = \boldsymbol{D} - \boldsymbol{A}$, with $\boldsymbol{D}$ being non negative diagonal and $\boldsymbol{A}$ being symmetric non negative. Further, assume that the eigenvalues of $\boldsymbol{D}^{-1}\boldsymbol{A}$ lie between $-\alpha$ and β. Then, $(1-\beta)\boldsymbol{D} \preceq \boldsymbol{D} - \boldsymbol{A} \preceq (1+\alpha)\boldsymbol{D}$.

Proof. See Appendix. $\qquad\square$

Combining the above results, give $\left[1 - \left(1 - \frac{1}{\kappa}\right)^{2^d}\right]\boldsymbol{D}_d \preceq \boldsymbol{D}_d - \boldsymbol{A}_d \preceq \left[1 + \left(1 - \frac{1}{\kappa}\right)^{2^d}\right]\boldsymbol{D}_d$ Hence, to guarantee that $\boldsymbol{D}_d \approx_{\epsilon_d} \boldsymbol{D}_d - \boldsymbol{A}_d$, the following system must be satisfied: (1) $e^{-\epsilon_d} \leq 1 - \left(1 - \frac{1}{\kappa}\right)^{2^d}$, and (2) $e^{\epsilon_d} \geq 1 + \left(1 - \frac{1}{\kappa}\right)^{2^d}$. Introducing γ for $\left(1 - \frac{1}{\kappa}\right)^{2^d}$, we arrive at: (1) $\epsilon_d \geq \ln\left(\frac{1}{1-\gamma}\right)$, and (2) $\epsilon_d \geq \ln(1+\gamma)$. Hence, $\epsilon_d \geq \max\left\{\ln\left(\frac{1}{1-\gamma}\right), \ln(1+\gamma)\right\} = \ln\left(\frac{1}{1-\gamma}\right)$. Now, notice that if $d = \lceil\log c\kappa\rceil$ then, $\gamma = \left(1 - \frac{1}{\kappa}\right)^{2^d} = \left(1 - \frac{1}{\kappa}\right)^{c\kappa} \leq \frac{1}{e^c}$. Hence, $\ln\left(\frac{1}{1-\gamma}\right) \leq \ln\left(\frac{e^c}{e^c-1}\right)$. This gives $c = \lceil 2\ln\left(\frac{\sqrt[3]{2}}{\sqrt[3]{2}-1}\right)\rceil$, implying $\epsilon_d = \ln\left(\frac{e^c}{e^c-1}\right) < \frac{1}{3}\ln 2$. $\qquad\square$

Using the above results the time complexity of DistRESolve with $d = \lceil\log\left(2\ln\left(\frac{\sqrt[3]{2}}{\sqrt[3]{2}-1}\right)\kappa\right)\rceil$ is $\mathcal{O}\left(n^2 \log\kappa \log(\frac{1}{\epsilon})\right)$ times steps, which concludes the proof of Theorem 1.

5. Distributed R-Hop SDDM Solver

Though the previous algorithm requires no knowledge of the graph's topology, but it requires the information of all other nodes (i.e., full communication). We will outline an R-Hop version of the algorithm in which communication is restricted to the R-Hop neighborhood between nodes. The following theorem summarizes these main results:

Theorem 2. *There is a decentralized algorithm* $\mathcal{A}(\{[\boldsymbol{M}_0]_{k1}, \dots [\boldsymbol{M}_0]_{kn}\}, [\boldsymbol{b}_0]_k, R, \epsilon)$, *that uses only R-Hop communication between the nodes and computes ϵ-close solutions to $\boldsymbol{M}_0\boldsymbol{x} = \boldsymbol{b}_0$ in* $\mathcal{O}\left(\left(\frac{\alpha\kappa}{R} + \alpha R d_{max}\right)\log(\frac{1}{\epsilon})\right)$ *time steps, with n being the number of nodes in $\mathbb{G}$, d_{max} denoting the maximal degree, κ the condition number of $\boldsymbol{M}_0$, and $\alpha = \min\left\{n, \frac{(d_{max}^{R+1}-1)}{(d_{max}-1)}\right\}$ representing the upper bound on the size of the R-hop neighborhood $\forall \boldsymbol{v} \in \mathbb{V}$, and $\epsilon \in (0, \frac{1}{2}]$ being the precision parameter.*

Given a graph $\mathbb{G}$ formed from the weighted Laplacian $\boldsymbol{M}_0$, the following corollary easily follows:

Corollary 1. *Let $\boldsymbol{M}_0$ be the weighted Laplacian of $\mathbb{G} = (\mathbb{V}, \mathbb{E}, \boldsymbol{W})$. There exists a decentralized algorithm that uses only R-hop communication between nodes and computes ϵ close solutions of $\boldsymbol{M}_0\boldsymbol{x} = \boldsymbol{b}_0$ in* $\mathcal{O}\left(\frac{n^3\alpha}{R}\frac{\boldsymbol{W}_{max}}{\boldsymbol{W}_{min}}\log(\frac{1}{\epsilon})\right)$ *time steps, with n being the number of nodes in $\mathbb{G}$, $\boldsymbol{W}_{max}, \boldsymbol{W}_{min}$ denoting the largest and the smallest weights of edges in $\mathbb{G}$, respectively, $\alpha = \min\left\{n, \frac{(d_{max}^{R+1}-1)}{(d_{max}-1)}\right\}$ representing the upper bound on the size of the R-hop neighborhood $\forall \boldsymbol{v} \in \mathbb{V}$, and $\epsilon \in (0, \frac{1}{2}]$ being the precision parameter.*

5.1. "Crude" R-Hop SDMM Solver

Algorithm 5 presents the "crude" R-Hop solver for SDDM systems using the same inverse chain of Section 4.1. Each node $\boldsymbol{v}_k \in \mathbb{V}$ receives the kth row of $\boldsymbol{M}_0$, k^{th} component, $[\boldsymbol{b}_0]_k$ of $\boldsymbol{b}_0$, the length of the inverse chain, d, and the local communication bound[2] R as inputs, and outputs the kth component of the "rude" approximation of $\boldsymbol{x}^\star$.

Analysis of Algorithm 5 The following Lemma shows that RDistRSolve computes the kth component of the "crude" approximation of $\boldsymbol{x}^\star$ and provides the algorithm's time complexity

---

[2]For simplicity, R is assumed to be in the order of powers of 2, i.e., $R = 2^p$.

Algorithm 5 RDistRSolve $(\{[\boldsymbol{M}_0]_{k1}, \ldots, [\boldsymbol{M}_0]_{kn}\}, [\boldsymbol{b}_0]_k, d, R)$

Part One:

$\{[\boldsymbol{A}_0\boldsymbol{D}_0^{-1}]_{k1}, \ldots, [\boldsymbol{A}_0\boldsymbol{D}_0^{-1}]_{kn}\} = \left\{\frac{[\boldsymbol{A}_0]_{k1}}{[\boldsymbol{D}_0]_{11}}, \ldots, \frac{[\boldsymbol{A}_0]_{kn}}{[\boldsymbol{D}_0]_{nn}}\right\}, \{[\boldsymbol{D}_0^{-1}\boldsymbol{A}_0]_{k1}, \ldots, [\boldsymbol{D}_0^{-1}\boldsymbol{A}_0]_{kn}\} = \{\frac{[\boldsymbol{A}_0]_{k1}}{[\boldsymbol{D}_0]_{kk}}, \ldots, \frac{[\boldsymbol{A}_0]_{kn}}{[\boldsymbol{D}_0]_{kk}}\}$

$[\boldsymbol{C}_0]_{k1}, \ldots, [\boldsymbol{C}_0]_{kn} = \text{Comp}_0([\boldsymbol{M}_0]_{k1}, \ldots, [\boldsymbol{M}_0]_{kn}, R), [\boldsymbol{C}_1]_{k1}, \ldots, [\boldsymbol{C}_1]_{kn} = \text{Comp}_1([\boldsymbol{M}_0]_{k1}, \ldots, [\boldsymbol{M}_0]_{kn}, R)$

Part Two:

for $i = 1$ to d **do**

    if $i - 1 < \rho$

    $[\boldsymbol{u}_1^{(i-1)}]_k = [\boldsymbol{A}_0\boldsymbol{D}_0^{-1}\boldsymbol{b}_{i-1}]_k$

    for $j = 2$ to 2^{i-1} **do**

        $[\boldsymbol{u}_j^{(i-1)}]_k = [\boldsymbol{A}_0\boldsymbol{D}_0^{-1}\boldsymbol{u}_{j-1}^{(i-1)}]_k$

    end for

    $[\boldsymbol{b}_i]_k = [\boldsymbol{b}_{i-1}]_k + [\boldsymbol{u}_{2^{i-1}}^{(i-1)}]_k$

    if $i - 1 \geq \rho$

    $l_{i-1} = 2^{i-1}/R$

    $[\boldsymbol{u}_1^{(i-1)}]_k = [\boldsymbol{C}_0\boldsymbol{b}_{i-1}]_k$

    for $j = 2$ to l_{i-1} **do**

        $[\boldsymbol{u}_j^{(i-1)}]_k = [\boldsymbol{C}_0\boldsymbol{u}_{j-1}^{(i-1)}]_k$

    end for

    $[\boldsymbol{b}_i]_k = [\boldsymbol{b}_{i-1}]_k + [\boldsymbol{u}_{l_{i-1}}^{(i-1)}]_k$

end for

Part Three:

$[\boldsymbol{x}_d]_k = [\boldsymbol{b}_d]_k/[\boldsymbol{D}_0]_{kk}$

for $i = d - 1$ to 1 **do**

    if $i < \rho$

    $[\boldsymbol{\eta}_1^{(i+1)}]_k = [\boldsymbol{D}_0^{-1}\boldsymbol{A}_0\boldsymbol{x}_{i+1}]_k$

    for $j = 2$ to 2^i **do**

        $[\boldsymbol{\eta}_j^{(i+1)}]_k = [\boldsymbol{D}_0^{-1}\boldsymbol{A}_0\boldsymbol{\eta}_{j-1}^{(i+1)}]_k$

    end for

    $[\boldsymbol{x}_i]_k = \frac{1}{2}\left[\frac{[\boldsymbol{b}_i]_k}{[\boldsymbol{D}_0]_{kk}} + [\boldsymbol{x}_{i+1}]_k + [\boldsymbol{\eta}_{2^i}^{i+1}]_k\right]$

    if $i \geq \rho$

    $l_i = 2^i/R$

    $[\boldsymbol{\eta}_1^{(i+1)}]_k = [\boldsymbol{C}_1\boldsymbol{x}_{i+1}]_k$

    for $j = 2$ to l_i **do**

        $[\boldsymbol{\eta}_j^{(i+1)}]_k = [\boldsymbol{C}_1\boldsymbol{\eta}_{j-1}^{(i+1)}]_k$

    end for

    $[\boldsymbol{x}_i]_k = \frac{1}{2}\left[\frac{[\boldsymbol{b}_i]_k}{[\boldsymbol{D}_0]_{kk}} + [\boldsymbol{x}_{i+1}]_k + [\boldsymbol{\eta}_{l_i}^{i+1}]_k\right]$

end for

$[\boldsymbol{x}_0]_k = \frac{1}{2}\left[\frac{[\boldsymbol{b}_0]_k}{[\boldsymbol{D}_0]_{kk}} + [\boldsymbol{x}_1]_k + [\boldsymbol{D}_0^{-1}\boldsymbol{A}_0\boldsymbol{x}_1]_k\right]$

return $[\boldsymbol{x}_0]_k$

Algorithm 6 $\text{Comp}_0([\boldsymbol{M}_0]_{k1}, \ldots, [\boldsymbol{M}_0]_{kn}, R)$

---

for $l = 1$ to $R - 1$ **do**

    for j s.t. $\boldsymbol{v}_j \in \mathbb{N}_{l+1}(\boldsymbol{v}_k)$ **do**

        $\left[(\boldsymbol{A}_0\boldsymbol{D}_0^{-1})^{l+1}\right]_{kj} = \sum_{r:\boldsymbol{v}_r \in \mathbb{N}_1(\boldsymbol{v}_j)} \frac{[\boldsymbol{D}_0]_{rr}}{[\boldsymbol{D}_0]_{jj}}[(\boldsymbol{A}_0\boldsymbol{D}_0^{-1})^l]_{kr}[\boldsymbol{A}_0\boldsymbol{D}_0^{-1}]_{jr}$

    end for

end for

return $\boldsymbol{c}_0 = \{[(\boldsymbol{A}_0\boldsymbol{D}_0^{-1})^R]_{k1}, \ldots, [(\boldsymbol{A}_0\boldsymbol{D}_0^{-1})^R]_{kn}\}$

Algorithm 7 $\text{Comp}_1([M_0]_{k1}, \ldots, [M_0]_{kn}, R)$

> **for** $l = 1$ to $R - 1$ **do**
>> **for** j s.t. $\boldsymbol{v}_j \in \mathbb{N}_{l+1}(\boldsymbol{v}_k)$ **do**
>>> $\left[(\boldsymbol{D}_0^{-1} \boldsymbol{A}_0)^{l+1} \right]_{kj} = \sum\limits_{r : \boldsymbol{v}_r \in \mathbb{N}_1(\boldsymbol{v}_j)} \frac{[\boldsymbol{D}_0]_{jj}}{[\boldsymbol{D}_0]_{rr}} [(\boldsymbol{D}_0^{-1} \boldsymbol{A}_0)^l]_{kr} [\boldsymbol{D}_0^{-1} \boldsymbol{A}_0]_{jr}$
>>
>> **end for**
>
> **end for**
> **return** $\boldsymbol{c}_1 = \{[(\boldsymbol{D}_0^{-1} \boldsymbol{A}_0)^R]_{k1}, \ldots, [(\boldsymbol{D}_0^{-1} \boldsymbol{A}_0)^R]_{kn}\}$

Lemma 11. *Let $\boldsymbol{M}_0 = \boldsymbol{D}_0 - \boldsymbol{A}_0$ be the standard splitting and let $\boldsymbol{Z}_0'$ be the operator defined by RDistRSolve, namely, $\boldsymbol{x}_0 = \boldsymbol{Z}_0' \boldsymbol{b}_0$. Then, $\boldsymbol{Z}_0' \approx_{\epsilon_d} \boldsymbol{M}_0^{-1}$. RDistRSolve requires $\mathcal{O}\left(\frac{2^d}{R}\alpha + \alpha R d_{max} \right)$, where $\alpha = \min\left\{ n, \frac{(d_{max}^{R+1} - 1)}{(d_{max} - 1)} \right\}$, to arrive at $\boldsymbol{x}_0$.*

The proof of the above Lemma can be arrived at by proving a collection of claims:

<u>Claim:</u> Matrices $\left(\boldsymbol{D}_0^{-1} \boldsymbol{A}_0 \right)^r$ and $\left(\boldsymbol{A}_0 \boldsymbol{D}_0^{-1} \right)^{-r}$ have sparsity patterns corresponding to the R-Hop neighborhood for any $R \in \mathbb{N}$.

Proof. The above claim is proved by induction on R. We start with the base case: for $R = 1$

$$[\boldsymbol{A}_0 \boldsymbol{D}_0^{-1}]_{ij} = \frac{[\boldsymbol{A}_0]_{ij}}{[\boldsymbol{D}_0]_{ii}} \quad \text{(if } j : \boldsymbol{v}_j \in \mathbb{N}_1(\boldsymbol{v}_i)) \qquad \text{or} \qquad [\boldsymbol{A}_0 \boldsymbol{D}_0^{-1}]_{ij} = 0 \quad \text{(otherwise)}$$

Therefore, $\boldsymbol{A}_0 \boldsymbol{D}_0^{-1}$ has a sparsity pattern corresponding to the 1-Hop neighborhood. Assume that for all $1 \leq p \leq R - 1$, $\left(\boldsymbol{A}_0 \boldsymbol{D}_0^{-1} \right)^p$ has a sparsity pattern corresponding to the $p - hop$ neighborhood. Consider, $\left(\boldsymbol{A}_0 \boldsymbol{D}_0^{-1} \right)^r$

$$[(\boldsymbol{A}_0 \boldsymbol{D}_0^{-1})^R]_{ij} = \sum_{k=1}^n [(\boldsymbol{A}_0 \boldsymbol{D}_0^{-1})^{R-1}]_{ik} [\boldsymbol{A}_0 \boldsymbol{D}_0^{-1}]_{kj} \tag{8}$$

Since $\boldsymbol{A}_0 \boldsymbol{D}_0^{-1}$ is non negative, then $[(\boldsymbol{A}_0 \boldsymbol{D}_0^{-1})^R]_{ij} \neq 0$ iff there exists k such that $\boldsymbol{v}_k \in \mathbb{N}_{R-1}(\boldsymbol{v}_i)$ and $\boldsymbol{v}_k \in \mathbb{N}_1(\boldsymbol{v}_j)$, namely, $\boldsymbol{v}_j \in \mathbb{N}_R(\boldsymbol{v}_i)$. The proof can be done in a similar fashion for $\boldsymbol{D}_0^{-1} \boldsymbol{A}_0$. $\qquad \square$

The next claim provides complexity guarantees for Comp_0 and Comp_1 described in Algorithms 6 and 7, respectively.

<u>Claim:</u> Algorithms 6 and 7 use only the R-hop information to compute the kth row of $\left(\boldsymbol{D}_0^{-1} \boldsymbol{A}_0 \right)^R$ and $\left(\boldsymbol{A}_0 \boldsymbol{D}_0^{-1} \right)^R$, respectively, in $\mathcal{O}\left(\alpha R d_{max} \right)$ time steps, where $\alpha = \min\left\{ n, \frac{(d_{max}^{R+1} - 1)}{(d_{max} - 1)} \right\}$.

Proof. The proof will be given for Comp_0 described in Algorithm 6 as that for Comp_1 can be performed similarly. Due to Claim 5.1, we have

$$\left[(\boldsymbol{A}_0 \boldsymbol{D}_0^{-1})^{l+1} \right]_{kj} = \sum_{r=1}^n \left[(\boldsymbol{A}_0 \boldsymbol{D}_0^{-1})^l \right]_{kr} \left[\boldsymbol{A}_0 \boldsymbol{D}_0^{-1} \right]_{rj} = \sum_{r : \boldsymbol{v}_r \in \mathbb{N}_1(\boldsymbol{v}_j)} \left[(\boldsymbol{A}_0 \boldsymbol{D}_0^{-1})^l \right]_{kr} \left[\boldsymbol{A}_0 \boldsymbol{D}_0^{-1} \right]_{rj} \tag{9}$$

Therefore at iteration $l + 1$, $\boldsymbol{v}_k$ computes the kth row of $\left(\boldsymbol{A}_0 \boldsymbol{D}_0^{-1} \right)^{l+1}$ using: (1) the kth row of $(\boldsymbol{A}_0 \boldsymbol{D}_0^{-1})^l$, and (2) the rth column of $\boldsymbol{A}_0 \boldsymbol{D}_0^{-1}$. Node $\boldsymbol{v}_r$, however, can only send the rth row of $\boldsymbol{A}_0 \boldsymbol{D}_0^{-1}$ making $\boldsymbol{A}_0 \boldsymbol{D}_0^{-1}$ non-symmetric. Noting that $[\boldsymbol{A}_0 \boldsymbol{D}_0^{-1}]_{rj} / [\boldsymbol{D}_0]_{rr} = [\boldsymbol{A}_0 \boldsymbol{D}_0^{-1}]_{jr} / [\boldsymbol{D}_0]_{jj}$, since $\boldsymbol{D}_0^{-1} \boldsymbol{A}_0 \boldsymbol{D}^{-1}$ is symmetric, leads to $[(\boldsymbol{A}_0 \boldsymbol{D}_0^{-1})^{l+1}]_{kj} = \sum\limits_{r : \boldsymbol{v}_r \in \mathbb{N}_1(\boldsymbol{v}_j)} \frac{[\boldsymbol{D}_0]_{rr}}{[\boldsymbol{D}_0]_{jj}} [(\boldsymbol{A}_0 \boldsymbol{D}_0^{-1})^l]_{kr} [\boldsymbol{A}_0 \boldsymbol{D}_0^{-1}]_{jr}$. To prove the time complexity guarantee, at each iteration $\boldsymbol{v}_k$ computes at most α values, where $\alpha = \min\left\{ n, \frac{(d_{max}^{R+1} - 1)}{(d_{max} - 1)} \right\}$ is the upper bound on the size of the R-hop neighborhood $\forall \boldsymbol{v} \in \mathbb{V}$. Each such computation requires at most $\mathcal{O}(d_{max})$ operations. Thus, the overall time complexity is given by $\mathcal{O}(\alpha R d_{max})$. $\qquad \square$

We are now ready to provide the proof of Lemma 11.

Proof. From **Parts Two** and **Three** of Algorithm 5, it is clear that node v_k computes $[b_1]_k, [b_2]_k, \ldots, [b_d]_k$ and $[x_d]_k, [x_{d-1}]_k, \ldots, [x_0]_k$, respectively. These are determined using the inverse approximated chain as follows

$$b_i = (I + (A_{i-1}D_{i-1}^{-1})b_{i-1} = b_{i-1} + (A_0 D_0^{-1})^{2^{i-1}} b_{i-1} \tag{10}$$

$$x_i = \frac{1}{2}[D_i^{-1}b_i + (I + D_i^{-1}A_i)x_{i+1}] = \frac{1}{2}[D_0^{-1}b_i + x_{i+1} + (D_0^{-1}A_0)^{2^i}x_{i+1}]$$

Considering the computation of $[b_1]_k, \ldots, [b_d]_k$ for $\rho > i - 1$, we have

$$[b_i]_k = [b_{i-1}]_k + [(A_0 D_0^{-1})^{2^{i-1}} b_{i-1}]_k = [b_{i-1}]_k + [\underbrace{A_0 D_0^{-1} \ldots A_0 D_0^{-1}}_{2^{i-1}} b_{i-1}]_k = [b_{i-1}]_k + [\underbrace{A_0 D_0^{-1} \ldots A_0 D_0^{-1}}_{2^{i-1}-1} u_1^{(i-1)}]_k \ldots$$

$$= [b_{i-1}]_k + \left[u_{2^{i-1}}^{(i-1)}\right]_k, \quad \text{with } u_{j+1}^{(i-1)} = A_0 D_0^{-1} u_j^{(i-1)} \text{ for } j = 1, \ldots 2^{i-1} - 1.$$

Since $A_0 D_0^{-1}$ has a sparsity pattern corresponding to 1-hop neighborhood (see Claim 5.1), node v_k computes $\left[u_{j+1}^{(i-1)}\right]_k$, based on $u_j^{(i-1)}$, acquired from its 1-Hop neighbors. It is easy to see that $\forall i$ such that $i - 1 < \rho$ the computation of $[b_i]_k$ requires $\mathcal{O}\left(2^{i-1}d_{max}\right)$ time steps. Thus, the computation of $[b_1]_k, \ldots, [b_\rho]_k$ requires $\mathcal{O}(2^\rho d_{max}) = \mathcal{O}(Rd_{max})$. Now, consider the computation of $[b_i]_k$ but for $i - 1 \geq \rho$

$$[b_i]_k = [b_{i-1}]_k + [(A_0 D_0^{-1})^{2^{i-1}} b_{i-1}]_k = [b_{i-1}]_k + [\underbrace{C_0 \ldots C_0}_{l_{i-1}} b_{i-1}]_k = [b_{i-1}]_k + [\underbrace{C_0 \ldots C_0}_{l_{i-1}-1} u_1^{(i-1)}]_k = [b_{i-1}]_k + \left[u_{l_{i-1}}^{(i-1)}\right]_k$$

with $C_0 = (A_0 D_0^{-1})^R$, $l_{i-1} = \frac{2^{i-1}}{R}$, and $u_{j+1}^{(i-1)} = C_0 u_j^{(i-1)}$ for $j = 1, \ldots, l_{i-1} - 1$. Since C_0 has a sparsity pattern corresponding to R-hop neighborhood (see Claim 5.1), node v_k computes $[u_{j+1}^{(i-1)}]_k$ based on the components of $u_j^{(i-1)}$ attained from its R-hop neighbors. For each i such that $i - 1 \geq \rho$ the computing $[b_i]_k$ requires $\mathcal{O}\left(\frac{2^{i-1}}{R}\alpha\right)$ time steps, where $\alpha = \min\left\{n, \frac{(d_{max}^{R+1}-1)}{(d_{max}-1)}\right\}$ being the upper bound on the number of nodes in the $R-$ hop neighborhood $\forall\, v \in \mathbb{V}$. Therefore, the overall computation of $[b_{\rho+1}]_k, [b_{\rho+2}]_k, \ldots, [b_d]_k$ is achieved in $\mathcal{O}\left(\frac{2^d}{R}\alpha\right)$ time steps. Finally, the time complexity for the computation of all of the values $[b_1]_k, [b_2]_k, \ldots, [b_d]_k$ is $\mathcal{O}\left(\frac{2^d}{R}\alpha + Rd_{max}\right)$. Similar analysis can be applied to determine the computational complexity of $[x_d]_k, [x_{d-1}]_k, \ldots, [x_1]_k$, i.e., **Part Three** of Algorithm 5. Using Lemma 5, we arrive at $Z_0' \approx_{\epsilon_d} M_0^{-1}$. Finally, using Claim 5.1, the time complexity of RDistRSolve (Algorithm 5) is $\mathcal{O}\left(\frac{2^d}{R}\alpha + \alpha Rd_{max}\right)$. $\square$

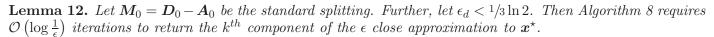### 5.2. "Exact" Distributed R-Hop SDDM Solver

Having developed an R-hop version which computes a "rude" approximation to the solution of $M_0 x = b_0$, now we provide an exact R-hop solver presented in Algorithm 8. Similar to RDistRSolve, each node v_k receives the kth row M_0, $[b_0]_k$, d, R, and a precision parameter ϵ as inputs, and outputs the kth component of the ϵ close approximation of vector $x^\star$.

Algorithm 8 EDistRSolve $(\{[M_0]_{k1}, \ldots, [M_0]_{kn}\}, [b_0]_k, d, R, \epsilon)$

Initialize: $[y_0]_k = 0$, and $[\chi]_k = \text{RDistRSolve}(\{[M_0]_{k1}, \ldots, [M_0]_{kn}\}, [b_0]_k, d, R)$
for $t = 1$ to q **do**
$[u_t^{(1)}]_k = [D_0]_{kk}[y_{t-1}]_k - \sum_{j:v_j \in \mathbb{N}_1(v_k)}[A_0]_{kj}[y_{t-1}]_j$
$[u_t^{(2)}]_k = \text{RDistRSolve}(\{[M_0]_{k1}, \ldots, [M_0]_{kn}\}, [u_t^{(1)}]_k, d, R)$
$[y_t]_k = [y_{t-1}]_k - [u_t^{(2)}]_k + [\chi]_k$
end for
return $[\tilde{x}]_k = [y_q]_k$

Analysis of Algorithm 8: The following Lemma shows that EDistRSolve computes the kth component of the ϵ close approximation to $x^\star$ and provides the time complexity analysis.

Lemma 12. *Let $M_0 = D_0 - A_0$ be the standard splitting. Further, let $\epsilon_d < 1/3\ln 2$. Then Algorithm 8 requires $\mathcal{O}\left(\log\frac{1}{\epsilon}\right)$ iterations to return the kth component of the ϵ close approximation to $x^\star$.*

Proof. See Appendix. $\square$

Next, the following Lemma provides the time complexity analysis of EDistRSolve.

Lemma 13. *Let $M_0 = D_0 - A_0$ be the standard splitting and let $\epsilon_d < 1/3\ln 2$, then EDistRSolve requires $\mathcal{O}\left((2^d/R\alpha + \alpha R d_{max})\log(1/\epsilon)\right)$ time steps. Moreover, for each node v_k, EDistRSolve only uses information from the R-hop neighbors.*

Proof. See Appendix $\square$

5.3. Length of the Inverse Chain

Again these introduced algorithms depend on the length of the inverse approximated chain, d. Here, we provide an analysis to determine the value of d which guarantees $\epsilon_d < \frac{1}{3}\ln 2$ in $\mathcal{C} = \{A_0, D_0, A_1, D_1, \ldots, A_d, D_d\}$. These results are summarized the following lemma

Lemma 14. *Let $M_0 = D_0 - A_0$ be the standard splitting and let κ denote the condition number M_0. Consider the inverse approximated chain $\mathcal{C} = \{A_0, D_0, A_1, D_1, \ldots, A_d, D_d\}$ with length $d = \left\lceil \log\left(2\ln\left(\frac{\sqrt[3]{2}}{\sqrt[3]{2}-1}\right)\kappa\right)\right\rceil$, then*

$$D_0 \approx_{\epsilon_d} D_0 - D_0\left(D_0^{-1}A_0\right)^{2^d}, \text{ with } \epsilon_d < 1/3\ln 2.$$

Proof. The proof is similar to that of Section 4.3 and can be found in the Appendix. $\square$

6. Discussion & Conclusions

We developed a distributed version of the parallel SDDM solver of [20] and proposed a fast decentralized solver for SDDM systems. Our approach is capable of acquiring ϵ-close solutions for arbitrary ϵ in $\mathcal{O}\left(n^3\frac{\alpha}{R}\frac{W_{max}}{W_{min}}\log\left(\frac{1}{\epsilon}\right)\right)$, with n the number of nodes in graph $\mathbb{G}$, W_{max} and W_{min} denoting the largest and smaller weights of the edges in $\mathbb{G}$, respectively, $\alpha = \min\left\{n, \frac{d_{max}^{R+1}-1}{d_{max}-1}\right\}$ representing the upper bound on the size of the R-Hop neighborhood $\forall v \in \mathbb{V}$, and $\epsilon \in (0, \frac{1}{2}]$ as the precision parameter. After developing the full communication version, we proposed a generalization to the R-Hop case where communication is restricted.

Our method is faster than state-of-the-art methods for iteratively solving linear systems. Typical linear methods, such as Jacobi iteration [2], are guaranteed to converge if the matrix is *strictly* diagonally dominant. We proposed a distributed algorithm that generalizes this setting, where it is guaranteed to converge in the SDD/SDDM scenario. Furthermore, the time complexity of linear techniques is $\mathcal{O}(n^{1+\beta}\log n)$, hence, a case of strictly diagonally dominant matrix M_0 can be easily constructed to lead to a complexity of $\mathcal{O}(n^4\log n)$. Consequently, our approach not only generalizes the assumptions made by linear methods, but is also faster by a factor of $\log n$.

In centralized solvers, nonlinear methods (e.g., conjugate gradient descent [11, 18], etc.) typically offer computational advantages over linear methods (e.g., Jacobi Iteration) for iteratively solving linear systems. These techniques, however, can not be easily decentralized. For instance, the stopping criteria for nonlinear methods require the computation of weighted norms of residuals (e.g., $||p_k||_{M_0}$ with p_k being the search direction at iteration k). To the best of our knowledge, the distributed computation of weighted norms is difficult. Namely using the approach in [19], this requires the calculation of the top singular value of M_0 which amounts to a power iteration on $M_0^\mathsf{T}M_0$ leading to the loss of sparsity. Furthermore, conjugate gradient methods require global computations of inner products.

Another existing method which we compare our results to is the recent work of the authors [16] where a local and asynchronous solution for solving systems of linear equations is considered. In their work, the authors derive a complexity bound, for one component of the solution vector, of $\mathcal{O}\left(\min\left(d\epsilon^{\frac{\ln d}{\ln ||G||_2}}, \frac{dn\ln\epsilon}{\ln ||G||_2}\right)\right)$, with ϵ being the precision parameter, d a constant bound on the maximal degree of $\mathbb{G}$, and G is defined as $x = Gx + z$ which can be directly mapped to $Ax = b$. The relevant scenario to our work is when A is PSD and G is symmetric. Here, the bound on the number of multiplications is given by $\mathcal{O}\left(\min\left(d^{\frac{\kappa(A)+1}{2}}\ln\frac{1}{\epsilon}, \frac{\kappa(A)+1}{2}nd\ln\frac{1}{\epsilon}\right)\right)$, with $\kappa(A)$ being the condition number of A. In the general case, when the degree depends on the number of nodes (i.e., $d = d(n)$), the minimum in the above bound will be the result of the second term ($\frac{\kappa(A)+1}{2}nd\ln\frac{1}{\epsilon}$) leading to $\mathcal{O}\left(d(n)n\kappa(A)\ln\frac{1}{\epsilon}\right)$. Consequently, in such a general setting, our approach outperforms [16] by a factor of $d(n)$.

References

[1] Efficient approximate solution of sparse linear systems. *Computers & Mathematics with Applications*, 1998.

[2] O. Axelsson. *Iterative Solution Methods*. Cambridge University Press, New York, NY, USA, 1994.

[3] J. Batson, D. A. Spielman, N. Srivastava, and S.-H. Teng. Spectral sparsification of graphs: Theory and algorithms. *Commun. ACM*, 56(8):87–94, Aug. 2013.

[4] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[5] G. E. Blelloch, A. Gupta, I. Koutis, G. L. Miller, R. Peng, and K. Tangwongsan. Near linear-work parallel SDD solvers, low-diameter decomposition, and low-stretch subgraphs. *CoRR*, abs/1111.1750, 2011.

[6] E. G. Boman and B. Hendrickson. Support Theory for Preconditioning. *SIAM J. Matrix Anal. Appl.*, 25(3):694–717, 2003.

[7] E. G. Boman, B. Hendrickson, and S. A. Vavasis. Solving elliptic finite element systems in near-linear time with support preconditioners. *CoRR*, cs.NA/0407022, 2004.

[8] P. Christiano, J. A. Kelner, A. Madry, D. A. Spielman, and S.-H. Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing*, STOC '11, pages 273–282, New York, NY, USA, 2011. ACM.

[9] S. I. Daitch and D. A. Spielman. Faster approximate lossy generalized flow via interior point algorithms. *CoRR*, abs/0803.0988, 2008.

[10] A. Joshi. Topics in optimization and sparse linear systems. Technical report, Champaign, IL, USA, 1996.

[11] E. Kaasschieter. Preconditioned conjugate gradients for solving singular systems. *Journal of Computational and Applied Mathematics*, 24(12):265 – 275, 1988.

[12] J. A. Kelner and A. Madry. Faster generation of random spanning trees. *CoRR*, abs/0908.1448, 2009.

[13] I. Koutis and G. L. Miller. A linear work, $o(n^{1/6})$ time, parallel algorithm for solving planar laplacians. pages 1002–1011, January 2007.

[14] I. Koutis, G. L. Miller, and R. Peng. Approaching optimality for solving SDD systems. *CoRR*, abs/1003.2958, 2010.

[15] I. Koutis, G. L. Miller, and R. Peng. Solving SDD linear systems in time $\tilde{O}(m \log n \log(1/\epsilon))$. *CoRR*, abs/1102.4842, 2011.

[16] C. E. Lee, A. E. Ozdaglar, and D. Shah. Solving systems of linear equations: Locally and asynchronously. *CoRR*, abs/1411.2647, 2014.

[17] J. Liu, S. Mou, and A. S. Morse. An asynchronous distributed algorithm for solving a linear algebraic equation. In *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*, pages 5409–5414. IEEE, 2013.

[18] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, 2nd edition, 2006.

[19] A. Olshevsky. Linear Time Average Consensus on Fixed Graphs and Implications for Decentralized Optimization and Multi-Agent Control. *ArXiv e-prints*, Nov. 2014.

[20] R. Peng and D. A. Spielman. An efficient parallel solver for SDD linear systems. *CoRR*, abs/1311.3286, 2013.

[21] D. A. Spielman and S. Teng. Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *CoRR*, abs/cs/0607105, 2006.

[22] D. A. Spielman and S. Teng. Spectral sparsification of graphs. *CoRR*, abs/0808.4134, 2008.

[23] X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *IN ICML*, pages 912–919, 2003.

Appendix

In this appendix, we will provide proof of the Lemmas in the original submission.

Lemma 1. *Let $\boldsymbol{Z}_0 \approx_\epsilon \boldsymbol{M}_0^{-1}$, and $\tilde{\boldsymbol{x}} = \boldsymbol{Z}_0 \boldsymbol{b}_0$. Then $\tilde{\boldsymbol{x}}$ is $\sqrt{2^\epsilon(e^\epsilon - 1)}$ approximate solution of $\boldsymbol{M}_0 \boldsymbol{x} = \boldsymbol{b}_0$.*

Proof. Let $\boldsymbol{x}^\star \in \mathbb{R}^n$ be the solution of $\boldsymbol{M}_0 \boldsymbol{x} = \boldsymbol{b}_0$, then

$$||\boldsymbol{x}^\star - \tilde{\boldsymbol{x}}||_{\boldsymbol{M}_0}^2 = (\boldsymbol{x}^\star - \tilde{\boldsymbol{x}})^T \boldsymbol{M}_0 (\boldsymbol{x}^\star - \tilde{\boldsymbol{x}}) = (\boldsymbol{x}^\star)^T \boldsymbol{M}_0 \boldsymbol{x}^\star + (\tilde{\boldsymbol{x}})^T \boldsymbol{M}_0 \tilde{\boldsymbol{x}} - 2(\boldsymbol{x}^\star)^T \boldsymbol{M}_0 \tilde{\boldsymbol{x}} \tag{11}$$

Consider each term separately in (11):

1. $(\boldsymbol{x}^\star)^T \boldsymbol{M}_0 \tilde{\boldsymbol{x}} = \boldsymbol{b}_0^T \boldsymbol{M}_0^{-1} \boldsymbol{M}_0 \boldsymbol{Z}_0 \boldsymbol{b}_0 = \boldsymbol{b}_0^T \boldsymbol{Z}_0 \boldsymbol{b}_0$
2. $(\boldsymbol{x}^\star)^T \boldsymbol{M}_0 \boldsymbol{x}^\star = \boldsymbol{b}_0^T \boldsymbol{M}_0^{-1} \boldsymbol{M}_0 \boldsymbol{M}_0^{-1} \boldsymbol{b}_0 = \boldsymbol{b}_0^T \boldsymbol{M}_0^{-1} \boldsymbol{b}_0 \leq e^\epsilon \boldsymbol{b}_0^T \boldsymbol{Z}_0 \boldsymbol{b}_0$
3. $\tilde{\boldsymbol{x}}^T \boldsymbol{M}_0 \tilde{\boldsymbol{x}} = \boldsymbol{b}_0^T \boldsymbol{Z}_0 \boldsymbol{M}_0 \boldsymbol{Z}_0 \boldsymbol{b}_0 \leq e^\epsilon \boldsymbol{b}_0^T \boldsymbol{Z}_0 \boldsymbol{b}_0$
   where in the last step we used that if $\boldsymbol{Z}_0 \approx_\epsilon \boldsymbol{M}_0^{-1}$, then $\boldsymbol{M}_0 \approx_\epsilon \boldsymbol{Z}_0^{-1}$.

Therefore, (11) can be rewritten as:

$$||\boldsymbol{x}^\star - \tilde{\boldsymbol{x}}||_{\boldsymbol{M}_0}^2 \leq 2(e^\epsilon - 1) \boldsymbol{b}_0^T \boldsymbol{Z}_0 \boldsymbol{b}_0 \tag{12}$$

Combining (12) with $\boldsymbol{b}_0^T \boldsymbol{Z}_0 \boldsymbol{b}_0 = (\boldsymbol{x}^\star)^T \boldsymbol{M}_0 \boldsymbol{Z}_0 \boldsymbol{M}_0 \boldsymbol{x}^\star \leq e^\epsilon (\boldsymbol{x}^\star)^T \boldsymbol{M}_0 \boldsymbol{x}^\star$:

$$||\boldsymbol{x}^\star - \tilde{\boldsymbol{x}}||_{\boldsymbol{M}_0}^2 \leq 2(e^\epsilon - 1) e^\epsilon (\boldsymbol{x}^\star)^T \boldsymbol{M}_0 \boldsymbol{x}^\star = 2(e^\epsilon - 1) e^\epsilon ||\boldsymbol{x}^\star||_{\boldsymbol{M}_0}^2$$

$\square$

Lemma 2. *Let $\boldsymbol{M}_0 = \boldsymbol{D}_0 - \boldsymbol{A}_0$ be the standard splitting of $\boldsymbol{M}_0$. Let $\boldsymbol{Z}_0'$ be the operator defined by DistrRSolve$([\{[\boldsymbol{M}_0]_{k1}, \ldots, [\boldsymbol{M}_0]_{kn}\}, [\boldsymbol{b}_0]_k, d)$ (i.e., $\boldsymbol{x}_0 = \boldsymbol{Z}_0' \boldsymbol{b}_0$). Then*

$$\boldsymbol{Z}_0' \approx_{\epsilon_d} \boldsymbol{M}_0^{-1}$$

Moreover, Algorithm 3 requires $\mathcal{O}\left(dn^2\right)$ time steps.

Proof. The proof commences by showing that $\left(\boldsymbol{D}_0^{-1} \boldsymbol{A}_0\right)^r$ and $\left(\boldsymbol{A}_0 \boldsymbol{D}_0^{-1}\right)^{-r}$ have a sparsity pattern corresponding to the r-hop neighborhood for any $r \in \mathbb{N}$. This case be shown using induction as follows

1. Base case: If $r = 1$, we have

$$[\boldsymbol{A}_0 \boldsymbol{D}_0^{-1}]_{ij} = \begin{cases} \frac{[\boldsymbol{A}_0]_{ij}}{[\boldsymbol{D}_0]_{ii}} & \text{if } j : \boldsymbol{v}_j \in \mathbb{N}_1(\boldsymbol{v}_i), \\ 0 & \text{otherwise .} \end{cases}$$

   Therefore, $\boldsymbol{A}_0 \boldsymbol{D}_0^{-1}$ has sparsity pattern corresponding to the 1-hop neighborhood.

Assume that for all $1 \leq p \leq r - 1$, $(\boldsymbol{A}_0 \boldsymbol{D}_0^{-1})^p$ has a sparsity patter corresponding to the p-hop neighborhood.

2. Now, consider $(\boldsymbol{A}_0 \boldsymbol{D}_0^{-1})^r$, where

$$[(\boldsymbol{A}_0 \boldsymbol{D}_0^{-1})^r]_{ij} = \sum_{k=1}^{n} [(\boldsymbol{A}_0 \boldsymbol{D}_0^{-1})^{r-1}]_{ik} [\boldsymbol{A}_0 \boldsymbol{D}_0^{-1}]_{kj} \tag{13}$$

   Since $\boldsymbol{A}_0 \boldsymbol{D}_0^{-1}$ is non negative then it is easy to see that $[(\boldsymbol{A}_0 \boldsymbol{D}_0^{-1})^r]_{ij} \neq 0$ if and only if there exists k such that $\boldsymbol{v}_k \in \mathbb{N}_{r-1}(\boldsymbol{v}_i)$ and $\boldsymbol{v}_k \in \mathbb{N}_1(\boldsymbol{v}_j)$ (i.e., $\boldsymbol{v}_j \in \mathbb{N}_r(\boldsymbol{v}_i)$).

For $\boldsymbol{D}_0^{-1}\boldsymbol{A}_0$, the same results can be derived similarly.

Please notice that in **Part One** of DistrRSolve algorithm node $\boldsymbol{v}_k$ computes (in a distributed fashion) the components $[\boldsymbol{b}_1]_k$ to $[\boldsymbol{b}_d]_k$ using the inverse approximated chain $\mathcal{C} = \{\boldsymbol{A}_0, \boldsymbol{D}_0, \boldsymbol{A}_1, \boldsymbol{D}_1, \ldots, \boldsymbol{A}_d, \boldsymbol{D}_d\}$. Formally,

$$\boldsymbol{b}_i = \left[\boldsymbol{I} + \left(\boldsymbol{A}_0\boldsymbol{D}_0^{-1}\right)^{2^{i-1}}\right]\boldsymbol{b}_{i-1} = \boldsymbol{b}_{i-1} + \left(\boldsymbol{A}_0\boldsymbol{D}_0^{-1}\right)^{2^{i-2}}$$

$$\left(\boldsymbol{A}_0\boldsymbol{D}_0^{-1}\right)^{2^{i-2}}\boldsymbol{b}_{i-1}$$

Clearly, at the ith iteration node $\boldsymbol{v}_k$ requires the kth row of $\left(\boldsymbol{A}_0\boldsymbol{D}_0^{-1}\right)^{2^{i-2}}$ (i.e., the kth row from the previous iteration) in addition to the jth row of $\left(\boldsymbol{A}_0\boldsymbol{D}_0^{-1}\right)^{2^{i-2}}$ from all nodes $\boldsymbol{v}_j \in \mathbb{N}_{2^{i-1}}(\boldsymbol{v}_k)$ to compute the kth row of $\left(\boldsymbol{A}_0\boldsymbol{D}_0^{-1}\right)^{2^{i-1}}$.

For computing $\left[\left(\boldsymbol{A}_0\boldsymbol{D}_0^{-1}\right)^{2^{i-1}}\right]_{kj}$, node $\boldsymbol{v}_k$ requires the kth row and j^{th} column of $\left(\boldsymbol{A}_0\boldsymbol{D}_0^{-1}\right)^{2^{i-2}}$. The problem, however, is that node $\boldsymbol{v}_j$ can only send the jth row of $\left(\boldsymbol{A}_0\boldsymbol{D}_0^{-1}\right)^{2^{i-2}}$ which can be easily seen not to be see that symmetric. To overcome this issue, node $\boldsymbol{v}_k$ has to compute the jth column of $\left(\boldsymbol{A}_0\boldsymbol{D}_0^{-1}\right)^{2^{i-2}}$ based on its jth row. The fact that $\boldsymbol{D}_0^{-1}\left(\boldsymbol{A}_0\boldsymbol{D}_0^{-1}\right)^{2^{i-2}}$ is symmetric, manifests that for $r = 1, \ldots, n$

$$\frac{\left[\left(\boldsymbol{A}_0\boldsymbol{D}_0^{-1}\right)^{2^{i-2}}\right]_{rj}}{[\boldsymbol{D}_0]_{rr}} = \frac{\left[\left(\boldsymbol{A}_0\boldsymbol{D}_0^{-1}\right)^{2^{i-2}}\right]_{jr}}{[\boldsymbol{D}_0]_{jj}}$$

Hence, for all $r = 1, \ldots, n$

$$\left[\left(\boldsymbol{A}_0\boldsymbol{D}_0^{-1}\right)^{2^{i-2}}\right]_{rj} = \frac{[\boldsymbol{D}_0]_{rr}}{[\boldsymbol{D}_0]_{jj}}\left[\left(\boldsymbol{A}_0\boldsymbol{D}_0^{-1}\right)^{2^{i-2}}\right]_{jr} \tag{14}$$

Now, lets analyze the time complexity of computing components $[\boldsymbol{b}_1]_k, [\boldsymbol{b}_2]_k, \ldots, [\boldsymbol{b}_d]_k$.

Time Complexity Analysis: At each iteration i, node $\boldsymbol{v}_k$ receives the jth row of $\left(\boldsymbol{A}_0\boldsymbol{D}_0^{-1}\right)^{2^{i-2}}$ from all nodes $\boldsymbol{v}_j \in \mathbb{N}_{2^{i-1}}(\boldsymbol{v}_k)$. using Equation 14, node $\boldsymbol{v}_k$ computes the corresponding columns as well as the product of these columns with the kth row of $\left(\boldsymbol{A}_0\boldsymbol{D}_0^{-1}\right)^{2^{i-2}}$. Therefore, the time complexity at the ith iteration is $\mathcal{O}\left(n^2 + \text{diam}(\mathbb{G})\right)$, where n^2 is responsible for the kth row computation, and $\text{diam}(\mathbb{G})$ represents the communication cost between the nodes. Using the fact that $\text{diam}(\mathbb{G}) \leq n$, the total complexity of **Part One** in DistrRSolve algorithm is $\mathcal{O}\left(dn^2\right)$.

In **Part Two**, node $\boldsymbol{v}_k$ computes (in a distributed fashion) $[\tilde{\boldsymbol{x}}_{d-1}]_k, [\tilde{\boldsymbol{x}}_{d-2}]_k, \ldots, [\tilde{\boldsymbol{x}}_0]_k$ using the same inverse approximated chain $\mathcal{C} = \{\boldsymbol{A}_0, \boldsymbol{D}_0, \boldsymbol{A}_1, \boldsymbol{D}_1, \ldots, \boldsymbol{A}_d, \boldsymbol{D}_d\}$.

$$\boldsymbol{x}_i = \frac{1}{2}\boldsymbol{D}_0^{-1}\boldsymbol{b}_i + \frac{1}{2}\left[\boldsymbol{I} + (\boldsymbol{D}_0^{-1}\boldsymbol{A}_0)^{2^i}\right]\boldsymbol{x}_{i+1} = \frac{1}{2}\boldsymbol{D}_0^{-1}\boldsymbol{b}_i + \frac{1}{2}\boldsymbol{x}_{i+1} \tag{15}$$

$$+ \frac{1}{2}\left(\boldsymbol{D}_0^{-1}\boldsymbol{A}_0\right)^{2^{i-1}}\left(\boldsymbol{D}_0^{-1}\boldsymbol{A}_0\right)^{2^{i-1}}\boldsymbol{x}_{i+1}$$

for $i = d - 1, \ldots, 1$. Thus,

$$\boldsymbol{x}_0 = \frac{1}{2}\boldsymbol{D}_0^{-1}\boldsymbol{b}_0 + \frac{\boldsymbol{x}_1}{2} + \frac{1}{2}\left(\boldsymbol{D}_0^{-1}\boldsymbol{A}_0\right)\boldsymbol{x}_1$$

Similar to the analysis of **Part One** of DistrRSolve algorithm the time complexity of **Part Two** as well as the time complexity of the whole algorithm is $\mathcal{O}\left(dn^2\right)$.

Finally, using Lemma 5 of the original paper for the inverse approximated chains $\mathcal{C} = \{\boldsymbol{A}_0, \boldsymbol{D}_0, \boldsymbol{A}_1, \boldsymbol{D}_1, \ldots, \boldsymbol{A}_d, \boldsymbol{D}_d\}$ yields:

$$\boldsymbol{Z}_0' \approx_{\epsilon_d} \boldsymbol{M}_0^{-1}.$$

$\square$

Lemma 3. *Let $\boldsymbol{M}_0 = \boldsymbol{D}_0 - \boldsymbol{A}_0$ be the standard splitting. Further, let $\epsilon_d < \frac{1}{3}\ln 2$ in the nverse approximated chain $\mathcal{C} = \{\boldsymbol{A}_0, \boldsymbol{D}_0, \boldsymbol{A}_1, \boldsymbol{D}_1, \ldots, \boldsymbol{A}_d, \boldsymbol{D}_d\}$. Then $DistrESolve(\{[\boldsymbol{M}_0]_{k1}, \ldots, [\boldsymbol{M}_0]_{kn}\}, [b_0]_k, d, \epsilon)$ requires $\mathcal{O}\left(\log\frac{1}{\epsilon}\right)$ iterations to return the kth component of the ϵ close approximation for $x^\star$.*

Proof. Notice that iterations in DistrESolve corresponds to Preconditioned Richardson Iteration:

$$\boldsymbol{y}_t = \left[\boldsymbol{I} - \boldsymbol{Z}_0' \boldsymbol{M}_0\right] \boldsymbol{y}_{t-1} + \boldsymbol{Z}_0 \boldsymbol{b}_0$$

where $\boldsymbol{Z}_0'$ is the operator defined by DistrRSolve and $\boldsymbol{y}_0 = \boldsymbol{0}$. Therefore, from Lemma 2:

$$\boldsymbol{Z}_0' \approx_{\epsilon_d} \boldsymbol{M}_0^{-1}$$

Finally, applying Lemma 6 of the main submission gives that DistrESolve algorithm needs $\mathcal{O}\left(\log \frac{1}{\epsilon}\right)$ iterations to k^{th} component of the ϵ approximated solution for $x^\star$. $\square$

Lemma 4. *Let $\boldsymbol{M}_0 = \boldsymbol{D}_0 - \boldsymbol{A}_0$ be the standard splitting. Further, let $\epsilon_d < \frac{1}{3}\ln 2$ in the inverse approximated chain $\mathcal{C} = \{\boldsymbol{A}_0, \boldsymbol{D}_0, \boldsymbol{A}_1, \boldsymbol{D}_1, \dots, \boldsymbol{A}_d, \boldsymbol{D}_d\}$. Then, DistrESolve($\{[\boldsymbol{M}_0]_{k1}, \dots, [\boldsymbol{M}_0]_{kn}\}, [\boldsymbol{b}_0]_k, d, \epsilon$) requires $\mathcal{O}\left(dn^2 \log(\frac{1}{\epsilon})\right)$ time steps.*

Proof. Each iteration of DistrESolve algorithm calls DistRSolve routine, therefore, using Lemmas 2 and 3 the total time complexity of f DistrESolve algorithm is $\mathcal{O}\left(dn^2 \log(\frac{1}{\epsilon})\right)$ time steps $\square$

<u>Claim:</u> Let κ be the condition number of $\boldsymbol{M}_0 = \boldsymbol{D}_0 - \boldsymbol{A}_0$, and $\{\lambda_i\}_{i=1}^n$ denote the eigenvalues of $\boldsymbol{D}_0^{-1}\boldsymbol{A}_0$. Then, $|\lambda_i| \leq 1 - \frac{1}{\kappa}$, for all $i = 1, \dots, n$

Proof. See Proposition 5.3 in [20]. $\square$

<u>Claim:</u> Let $\boldsymbol{M}$ be an SDDM matrix and consider the splitting $\boldsymbol{M} = \boldsymbol{D} - \boldsymbol{A}$, with $\boldsymbol{D}$ being non negative diagonal and $\boldsymbol{A}$ being symmetric non negative. Further, assume that the eigenvalues of $\boldsymbol{D}^{-1}\boldsymbol{A}$ lie between $-\alpha$ and β. Then, $(1 - \beta)\boldsymbol{D} \preceq \boldsymbol{D} - \boldsymbol{A} \preceq (1 + \alpha)\boldsymbol{D}$.

Proof. See Proposition 5.4 in [20]. $\square$

Lemma 5. *Let $\boldsymbol{M}_0 = \boldsymbol{D}_0 - \boldsymbol{A}_0$ be the standard splitting. Further, let $\epsilon_d < 1/3 \ln 2$. Then Algorithm 8 requires $\mathcal{O}\left(\log \frac{1}{\epsilon}\right)$ iterations to return the kth component of the ϵ close approximation to $\boldsymbol{x}^\star$.*

Proof. Please note that the iterations of EDistRSolve correspond to a distributed version of the preconditioned Richardson iteration scheme

$$\boldsymbol{y}_t = [\boldsymbol{I} - \boldsymbol{Z}_0' \boldsymbol{M}_0]\boldsymbol{y}_{t-1} + \boldsymbol{Z}_0' \boldsymbol{b}_0$$

with $\boldsymbol{y}_0 = 0$ and $\boldsymbol{Z}_0'$ being the operator defined by RDistRSolve. From Lemma 11 it is clear that $\boldsymbol{Z}_0' \approx_{\epsilon_d} \boldsymbol{M}_0^{-1}$. Applying Lemma 6, provides that EDistRSolve requires $\mathcal{O}(\log 1/\epsilon)$ iterations to return the kth component of the ϵ close approximation to $\boldsymbol{x}^\star$. Finally, since EDistRSolve uses procedure RDistRSolve as a subroutine, it follows that for each node $\boldsymbol{v}_k$ only communication between the R-hope neighbors is allowed. $\square$

Lemma 6. *Let $\boldsymbol{M}_0 = \boldsymbol{D}_0 - \boldsymbol{A}_0$ be the standard splitting and let $\epsilon_d < 1/3 \ln 2$, then EDistRSolve requires $\mathcal{O}\left((2^d/R\alpha + \alpha R d_{max}) \log(1/\epsilon)\right)$ time steps. Moreover, for each node $\boldsymbol{v}_k$, EDistRSolve only uses information from the R-hop neighbors.*

Proof. Notice that at each iteration EDistRSolve calls RDistRSolve as a subroutine, therefore, for each node v_k only R-hop communication is allowed. Lemma 11 gives that the time complexity of each iteration is $\mathcal{O}\left(\frac{2^d}{R}\alpha + \alpha R d_{max}\right)$, and using Lemma 5 immediately gives that the time complexity of $\mathcal{O}\left((2^d/R\alpha + \alpha R d_{max}) \log(1/\epsilon)\right)$. $\square$

Lemma 7. *Let $\boldsymbol{M}_0 = \boldsymbol{D}_0 - \boldsymbol{A}_0$ be the standard splitting and let κ denote the condition number $\boldsymbol{M}_0$. Consider the inverse approximated chain $\mathcal{C} = \{\boldsymbol{A}_0, \boldsymbol{D}_0, \boldsymbol{A}_1, \boldsymbol{D}_1, \dots, \boldsymbol{A}_d, \boldsymbol{D}_d\}$ with length $d = \lceil \log\left(2\ln\left(\frac{\sqrt[3]{2}}{\sqrt[3]{2}-1}\right)\kappa\right)\rceil$, then $\boldsymbol{D}_0 \approx_{\epsilon_d} \boldsymbol{D}_0 - \boldsymbol{D}_0 \left(\boldsymbol{D}_0^{-1}\boldsymbol{A}_0\right)^{2^d}$, with $\epsilon_d < 1/3 \ln 2$.*

Proof. The proof takes similar steps as in Lemma 14. $\square$